Efficient Vertex-Label Distance Oracles for Planar Graphs ⋆

Shay Mozes, Eyal E. Skop

Efi Arazi School of Computer Science
The Interdisciplinary Center Herzliya

Abstract. We consider distance queries in vertex labeled planar graphs. For any fixed $0 < \epsilon \leq 1/2$ we show how to preprocess an undirected planar graph with vertex labels and edge lengths to answer queries of the following form. Given a vertex u and a label λ return a $(1+\epsilon)$-approximation of the distance between u and its closest vertex with label λ. The preprocessing time is $O(\epsilon^{-2} n \lg^3 n)$, the required space is $O(\epsilon^{-1} n \lg n)$, and the query time is $O(\lg\lg n + \epsilon^{-1})$. For a directed planar graph with arc lengths bounded by N, the preprocessing time is $O(\epsilon^{-2} n \lg^3 n \lg(nN))$, the space is $O(\epsilon^{-1} n \lg n \lg(nN))$, and the query time is $O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})$.

1 Introduction

Imagine you are driving your car and suddenly see you are about to run out of gas. What should you do? Obviously, you should find the closest gas station. This is the *vertex-to-label distance query problem.* Various software applications like Waze and Google maps attempt to provide such a functionality. The idea is to preprocess the locations of service providers, such as gas stations, hospitals, pubs and metro stations in advance, so that when a user, whose location is not known a priori, asks for the distance to the closest service provider, the information can be retrieved as quickly as possible.

We study this problem from a theoretical point of view. We model the network as a planar graph with labeled vertices (e.g., a vertex labeled as a gas station). We study distance oracles for such graphs. A *vertex-label distance oracle* is a data structure that represents the input graph and can be queried for the distance between any vertex and the closest vertex with a desired label. We consider approximate distance oracles, which, for any given fixed parameter $\epsilon > 0$, return a distance estimate that is at least the true distance queried, and at most $(1+\epsilon)$ times the true distance (this is known as a $(1+\epsilon)$-stretch). One would like an oracle with the following properties; queries should be answered quickly, the oracle should consume little space, and the construction of the oracle should take as little time as possible. We use the notation $\langle O(S(n))_{space}\ , O(T(n))_{time}\rangle$ to express the space requirement and query time of a distance oracle.

The vertex-to-label distance query problem was introduced by Hermelin, Levy, Weimann and Yuster [6]. For any integer $k \geq 2$, they gave a $(4k -$

⋆ This work was partially supported by Israel Science Foundation grant 794/13.

5)-stretch $\langle O(kn^{1+1/k})_{space}\ ,\ O(k)_{time}\rangle$ vertex-label distance oracle (expected space) for undirected general (i.e., non-planar) graphs. This is not efficient when the number l of distinct labels is $o(n^{1/k})$. They also presented a (2^k-1)-stretch $\langle O(knl^{1/k})_{space}\ ,\ O(k)_{time}\rangle$ undirected oracle, and showed how to maintain label changes in sub-linear time. Chechik [4] improved the latter two results to $(4k-5)$-stretch and similar space/time bounds.

For planar graphs, the only vertex-label distance oracle we are aware of was described by Li, Ma and Ning [10]. They construct a $(1+\epsilon)$-stretch oracle with $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\epsilon^{-1} \lg n \lg \rho)_{time}\rangle$ bounds for undirected graphs. Here, ρ is the radius of the graph, which can be $\theta(n)$. It is also shown in [10] how to avoid the $\lg \rho$ factor when $\rho = O(\lg n)$. The construction time of their oracle is $O(\epsilon^{-1} n \log^2 n)$.

Our results and approach We give a $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ ,\ O(\lg\lg n + \epsilon^{-1})_{time}\rangle$ vertex-label distance oracle for *undirected* planar graphs that can be constructed in $O(\epsilon^{-2} n \lg^3 n)$ time. This improves over the query time of [10] by roughly a $\log^2 n$ factor. For directed planar graphs we give a $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}\ ,\ O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ vertex-label distance oracle whose construction time is $O(\epsilon^{-2} n \lg^3 n \lg(nN))$. To the best of our knowledge, no non-trivial directed vertex-label distance oracles were proposed prior to the current work.

Consider a vertex-to-vertex distance oracle for a graph with label set L. If the oracle works for general directed graphs then the vertex-to-label problem can be solved easily; add a distinct apex v_λ for each label $\lambda \in L$, and connect every λ-labeled vertex to v_λ with a zero length arc. Finding the distance from a vertex u to label λ is now equivalent to finding the distance between u and v_λ. This approach presents two main difficulties when designing efficient oracles for planar graphs. First, adding apices breaks planarity. In particular, it affects the separability of the graph. Thus, the reduction does not work with oracles that depend on planarity or on the existence of separators, which are more efficient than oracles for general graphs. Second, the reduction uses *directed* arcs, so it is unsuitable for oracles for undirected graphs. Using arcs in the reduction is crucial since connecting an apex with undirected zero length edges changes the distances in the graph. This is because the apex can be used to teleport between vertices with the same label.[1]

We nonetheless use this approach, and show how to overcome these obstacles. We augment a directed and an undirected variants of a distance oracle of Thorup [12] for planar graphs. These oracles rely on the existence of fundamental cycle separators in planar graphs, a property that breaks when apices are added to the graph. However, we observe that once the graph is separated, Thorup's oracle does not depend on planarity. We therefore postpone the addition of the apices till a later stage in the construction of the distance oracle, when

[1] Teleporting between vertices might be desirable in some applications. For example, calculating the walking distance between two locations without accounting traveling between train stations.

the graph has already been separated. We show that, nonetheless, approximate distances from any vertex to any label in the entire graph can be approximated. Moreover, we observe that Thorup's undirected oracle internally uses the same directed structures as in his directed oracle. It only depends on the undirectedness in making the number and sizes of these structures smaller than in the directed case. We extend this argument to handle vertex labels.

Additional Related Work We summarize related work on approximate vertex-vertex distance oracles. For general graphs, no efficient (2)-stretch approximate vertex-vertex distance oracles are known to date. Thorup and Zwick [13] presented a $(2k-1)$-stretch $\langle O(kn^{1+1/k})_{space}, O(k)_{time}\rangle$ undirected distance oracle which is constructed in $O(kmn^{1/k})$ time. Wulff-Nilsen [14] achieved the same result with preprocessing of $O(kn^{1+\frac{c}{k}})$ for universal constant c. Several more improvements of [13] have been found for unweighted or sparse graphs ([1], [2], [3]).

In contrast, vertex-vertex oracles for planar graphs with stretch less then 2 have been constructed. Thorup [12] gave a $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}, O(\lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ stretch $(1+\epsilon)$ directed distance oracle, and a $\langle O(\epsilon^{-1} n \lg n)_{space}, O(\epsilon^{-1})_{time}\rangle$ undirected (simplified) distance oracle. Our result is base on Thorup's oracles, which are described in Section 3. Klein [9] independently gave an undirected distance oracle with same bounds. Kawarabayashi, Klein and Sommer [7] have shown a $\langle O(n)_{space}, O(\epsilon^{-2} \lg^{-2}(n))_{time}\rangle$ undirected $(1+\epsilon)$-stretch distance oracle constructed in $O(n \lg^2 n)$ time, inspired by [12]. [7] give a trade-off of $\langle O(\frac{\epsilon^{-1} n \lg n}{\sqrt{r}})_{space}, O(r + \sqrt{r}\epsilon^{-1} \lg n)_{time}\rangle$ oracle algorithms. Kawarabayashi et al. [8] have shown better tradeoffs for undirected oracles. For the case where $N \in poly(n)$, they achieve $\langle O^*(n \lg n)_{space}, O^*(\epsilon^{-1})_{time}\rangle$ oracle, where O^* hides $\lg(\epsilon^{-1})$ and $\lg^*(n)$ factors.

Roadmap In this extended abstract we focus on the undirected case. The remainder of this paper is organized as follows. We describe the scheme of the vertex-to-vertex distance oracle of Thorup in Section 3. In Section 4, we describe a vertex-labeled oracle for undirected planar graphs. Our description goes into some of details that cannot be cited and used from [12] due to a minor flaw in the treatment of the undirected case in [12]. In Appendix B we elaborate on the flaw in [12] and explain how to it is corrected. Due to space constraints, our vertex-labeled distance oracle for directed planar graphs is described in Appendix C. Its construction is similar to the undirected oracle, but relies on some additional reductions from [12] that we use without change.

2 Preliminaries

Let $V(G)$ denote the vertex set of a graph G. We use the terms arcs and edges to distinguish directed and undirected graphs. Let $A(G)$ ($E(G)$) denote the arc (edge) set of a directed (undirected) graph. We denote the concatenation of two paths P_1 and P_2 that share an endpoint by $P_1 \circ P_2$.

For a simple path Q and a vertex set $U \subseteq V(Q)$, we define $\bar{Q}$, the *reduction* of Q to U as follows. Repeatedly apply the following procedure to Q. Let wv be an edge of Q s.t. $v \notin U$. Contract wv, and add the length of wv to the length of the other edge of Q incident to w, if such one exists. Note that $|V(\bar{Q})| = O(|U|)$.

Let T be a rooted spanning tree of a graph G. For $u \in V(G)$, let $T[u]$ denote the unique root-to-u path in T. The *fundamental cycle* of $e = (u_1, u_2) \notin E(T)$ is the undirected cycle composed of $E(T[u_1]), E(T[u_2])$, and e.

Let $L = \{\lambda_i\}_{i=1}^{l}$ be a set of l labels. A vertex-labeled graph is a graph $G = (V, A)$, equipped with a function $f : V \to L$. Let $V_\lambda = \{v \in V(G) | f(v) = \lambda\}$ to be set of vertices with label λ.

Let G be a graph with arc lengths. For $u, v \in V(G)$, let $\delta_G(u, v)$ denote the u-to-v distance in G. For a vertex-labeled G, we define $\delta_G(u, \lambda) = \min_{w \in V_\lambda} \delta_G(u, w)$.

We assume basic familiarity with planar graphs. In particular, it is well known that if G is planar then $|A(G)| = O(|V(G)|)$, and that a simple cycle separates a planar graph G into an interior and an exterior parts.

A *vertex-label distance oracle* is a data structure that, given a vertex $v \in V$ and a label $\lambda \in L$, outputs an (approximation) of $\delta_G(v, \lambda)$. We note that this problem is a generalization of the basic distance oracle problem in which each vertex is given a unique label. Constructing an $O(nl)$-space vertex-label distance oracle is trivial. Simply precompute and store the distance between each vertex and each possible label. The goal is, therefore, to devise an oracle which requires substantially less than nl space, while allowing for fast queries.

3 Thorup's Approximate Distance Oracle

In this section we outline the distance oracle of Thorup [12]. This is necessary for understanding our results. The oracle we describe differs from the original in [12] in some of the details. See Appendix B for an explanation of the differences.

3.1 ϵ-covering sets

The main idea is to store just a subset of the pairwise distances in the graph, from which all distances can be approximately computed efficiently. Given an undirected graph H, and a shortest path $Q \in H$, Thorup shows that for every vertex $v \in H$, there exists a set of $O(\epsilon^{-1})$ vertices on Q, called *connections*, such that the distances (called *connection lengths*) between every vertex of H and its connections on Q can be used to approximate, in $O(\epsilon^{-1})$ time, the length of any shortest path in H that intersects Q. Thorup essentially proves the following: [2]

Lemma 1. *Let Q be a shortest path in an undirected graph H. There exist sets $C(u, Q)$ of $O(\epsilon^{-1})$ vertices of Q for all $u \in H$, where:*

1. *$C(u, Q)$ are called the* connections *of u on Q.*

[2] While Lemma 1 true, we believe there is a flaw in the arguments in [12], see Appendix B.

2. *The distance between* u *and a connection* $q \in C(u,Q)$ *is called the* connection length of u and q.
3. *For every* $u,w \in H$*, if a shortest* u*-to-*w *path in* H *intersects* Q*, then* $\delta_{H_Q^{uw}}(u,w) \le (1+\epsilon)\delta_H(u,w)$.
   Here H_Q^{uw} *is the graph with vertices* u,w*, and the vertices of the reduction of* Q *to* $C(u,Q) \cup C(w,Q)$*, and with* u*-to-*Q *and* Q*-to-*w *edges whose lengths are the corresponding connection lengths of* $C(u,Q)$ *and* $C(w,Q)$*.*

Note that, since for every v $|C(v,Q)| = O(\epsilon^{-1})$, computing $\delta_{H_Q^{uw}}(u,w)$ can be done in time that only depends on ϵ^{-1} (in fact in $O(\epsilon^{-1})$ time).

In the remainder of this subsection we prove Lemma 1.

For efficiency reasons, instead of storing exact connection lengths $\delta(\cdot,\cdot)$, the algorithm computes approximate connection lengths, which we denote by $\ell(\cdot,\cdot)$.

This following definition captures the intuitive idea that if a v-to-q path that goes through q^* is not too much longer than the shortest v-to-q path, then it suffices to store the distance from v to q^* and the distance from q^* to q.

Definition 1. q^* ϵ-covers q *w.r.t.* v *if* $\ell(v,q^*) + \delta(q^*,q) \le (1+\epsilon)\delta(v,q)$*.*

Thorup [12] uses a different notion of covering.[3]

Definition 2. q^* quasi ϵ-covers q *w.r.t.* v *if* $\ell(v,q^*)+\delta(q^*,q) \le \delta(v,q)+\epsilon\ell(v,q^*)$*.*

Let Q be a path. A set C of vertices of Q is a *(quasi)-*ϵ*-covering* of Q w.r.t. v if for every $q \in Q$ there is a connection $q^* \in C$ that (quasi)-ϵ-covers q w.r.t. v.

A covering set is called *clean* if it is inclusion-wise minimal and *ordered* if it is sorted by the order of connections along the path Q. Observe that by keeping the distance of every $q \in Q$ from the first vertex of Q, allows computing $\delta_Q(q,q')$ for any $q,q' \in Q$ in constant time.

The notions of ϵ-covering sets and quasi-ϵ-covering sets are related by the following proposition:

Proposition 1. *Let* $C(v,Q)$ *be a quasi* ϵ*-covering set. For any* $0 < \epsilon \le 1/2$*,* $C(v,Q)$ *is a* 2ϵ*-covering set.*

Proof. If q^* quasi ϵ-covers q then $\ell(q^*,v) \le \frac{1}{1-\epsilon}\delta(q,v) \le 2\delta(q,v)$. Hence $\delta(q,q^*)+\ell(q^*,v) \le \delta(q,v) + \epsilon\ell(q^*,v) \le (1+2\epsilon)\delta(q,v)$. Therefore, if $C(v,Q)$ is a quasi ϵ-covering set, it is a 2ϵ-covering set. ☐

The following lemma shows that, in order to prove Lemma 1, it is suffices that the sets $C(v,Q)$ be ϵ-covering sets of size $O(\epsilon^{-1})$.

Lemma 2. *([9, Lemma 4.1]*[4]*) Let* u,w *be vertices in an undirected graph* H*. Let* Q *be a shortest path in* H *such that a* u*-to-*w *shortest path intersects* Q*. Let* $C(u,Q), C(w,Q)$ *be* ϵ*-covering sets of* Q *w.r.t.* u,w*, respectively. Let* H_Q^{uw} *be as in the statement of Lemma 1. Then,*

$$\delta_{H_Q^{uw}}(u,w) \le (1+\epsilon)\delta_H(u,w) \tag{1}$$

[3] The term quasi-ϵ-cover is not used by Thorup. He uses ϵ-covers for this notion.
[4] Klein showed this lemma for ϵ-covering sets, while Thorup showed a similar lemma using a different notion of ϵ-covering sets.

Thorup shows how to efficiently construct quasi-ϵ-covering sets. Let Q be a shortest path in an undirected graph H. Let $sssp(Q,H)$ be the smallest number s.t. for any subgraph H_0 of H, and any vertex $q \in Q_0$, where Q_0 is the reduction of Q to H_0, we can compute single source shortest paths from q in the graph $Q_0 \cup H_0$ in $O(sssp(Q,H)|E(H_0)|)$ time. It is easy to see that a standard implementation of Dijkstra's algorithm with priority queues implies $sssp(Q,H) = O(\lg |E(H)|)$. If H is planar, then $sssp(Q,H) = O(1)$ by [5].

Lemma 3. *([12, Lemma 3.18]) Given an undirected graph H and shortest path Q, quasi ϵ-covering sets of Q with respect to all vertices of H, each of size $O(\epsilon^{-1}\lg n)$, can be constructed in $O(\epsilon^{-1} sssp(Q,H)|E(H)|\lg(|V(Q)|))$ time.*

By Proposition 1 the quasi ϵ-covers produced by Lemma 3 are 2ϵ-covering sets. However, their sizes are too large. The sizes can be decreased using the following thinning procedure. The proof appears in Appendix A.[5]

Lemma 4. *Let Q be a path in an undirected graph, and let v be a vertex. Let $D(v,Q)$ be an ordered ϵ_0-cover of Q w.r.t. v. For any $\epsilon_1 \leq 1$, a clean and ordered $(2\epsilon_0+\epsilon_1)$-cover $C(v,Q) \subseteq D(v,Q)$ of size $O(\epsilon_1^{-1})$ can be constructed in $O(|D(v,Q)|)$ time.*

Thus, by combining Lemma 3, Proposition 1, and Lemma 4, we get the following corollary, which, along with Lemma 2, establishes Lemma 1.

Corollary 1. *Given an undirected graph H and a shortest path Q, ϵ-covering sets of Q with respect to all vertices of H, each of size $O(\epsilon^{-1})$, can be constructed in $O(\epsilon^{-1} sssp(Q,H)|E(H)|\lg(|V(Q)|))$ time.*

3.2 The distance oracle

The construction is recursive, using shortest path separators.

Lemma 5. *(Fundamental Cycle Separator [11]) Let H be an undirected planar graph with a rooted spanning tree T and function w assigning non-negative weights to edges. One can find an edge $e \notin T$ such that neither the weight strictly enclosed by the fundamental cycle of e nor the weight not enclosed by the fundamental cycle of e exceeds $\frac{2}{3}$ the weight of H.*

A planar graph G can be decomposed by computing a shortest path tree for an arbitrary vertex, and applying Lemma 5 recursively. Choosing the spanning tree in Lemma 5 to be a shortest path tree guarantees that each fundamental cycle separator found consists of two shortest paths. The decomposition can be represented by a binary tree $\mathcal{T}$ in the following manner. [6] See Figure 1 in the appendix for an illustration.

[5] In [12] a thinning procedure is given only for the directed case, and it is claimed that a quasi-ϵ-covering set can be thinned. We believe this is not correct. See Appendix B. Instead, we give here a thinning procedure for ϵ-covering sets (not quasi-ϵ-covering).

[6] We refer to the vertices of $\mathcal{T}$ as *nodes* to distinguish them from the vertices of the graph G.

– Each node r of $\mathcal{T}$ is associated with a subgraph G_r of G. The subgraph associated with the root of $\mathcal{T}$ is all of G.
– Each non-leaf node r of $\mathcal{T}$ is associated with the fundamental cycle separator S_r found by invoking Lemma 5 on G_r.
– Each non-leaf node r has two children, whose associated subgraphs are the interior and exterior of S_r. The vertices and edges of the separator belong to both subgraphs.

Let r be a node of $\mathcal{T}$. The *frame* F_r of G_r is the set of (shortest) paths in $\bigcup_{r'}(S_{r'} \cap G_r)$, where the union is over strict ancestors r' of r in $\mathcal{T}$. Each non-leaf node r stores its frame F_r. A standard argument shows that, by alternating the separation criteria between number of edges in the graph and number of paths in the frame, one can get frames consisting of a constant number of paths.

For $r \in \mathcal{T}$, let G_r° denote the subgraph of $G_r \setminus F_r$. That is, G_r° is the graph obtained from G_r by removing the edges of the frame F_r as well as any vertices of F_r that become isolated as a result of the removal. The sizes of the G_r°'s decrease by a constant factor along $\mathcal{T}$, while the sizes of the G_r's need not because there is no bound on the size of the fundamental cycle in Lemma 5. This may pose a problem, since the frame F_r is stored by every node r. To overcome this, the algorithm stores the reduction of F_r to G_r° instead of F_r itself.

Let u, w be vertices of G. Let r_u, r_w be the leaves of $\mathcal{T}$ such that $u \in G_{r_u}$ and $w \in G_{r_w}$. Let r be the LCA of r_u and r_v in $\mathcal{T}$. Observe that u and w are separated by S_r. Hence, every u-to-w path in G must intersect S_r. However, a u-to-w path may or may not intersect F_r. See Figure 2 in the appendix. Suppose first that a shortest u-to-w path P (in G) does intersect F_r. We write $P = P_0 \circ P_1$. Path P_0 is a maximal prefix of P whose vertices belong to G_r°. We call this kind of paths *type-0* paths. Note that type-0 paths start at a vertex of G_r°, end at a vertex of F_r and are confined to G_r°. Path P_1 consists of the remainder of P, and is referred to as a *type-1* path. Note that type-1 paths start at a vertex of $F_r \cap G_r^\circ$, end at a vertex of G_r°, but are *not* confined to G_r°. It is not difficult to convince oneself that, to be able to approximate $\delta_G(u, w)$, it suffices to keep, for every $Q \in F_r$, connections $C(u, Q)$ of type-0 (i.e. the connection lengths are relative to G_r°, not the entire G) and connections $C(w, Q)$ of type-1 (i.e. the connection lengths are relative to the entire graph G).

Now suppose that no shortest u-to-w path P (in G) intersects F_r. Then every u-to-w path P (in G) intersects S_r and is confined to G_r°. Then, to approximate P it suffices to keep, for every $Q \in S_r$, type-0 connections $C(u, Q)$ and $C(w, Q)$.

The distance oracle therefore keeps, for each $r \in \mathcal{T}$, for each vertex $u \in G_r^\circ$:

1. connections $C(u, Q)$ of type-0 for all $Q \in F_r$.
2. connections $C(u, Q)$ of type-1 for all $Q \in F_r$.
3. connections $C(u, Q)$ of type-0 for all $Q \in S_r$.

These connections, over all $u \in G_r$ and all paths in $F_r \cup S_r$ are called the (type-0 or type-1) connections of r. In addition, the data structure stores:

– A mapping of each vertex $v \in G$ to a leaf node $r_v \in \mathcal{T}$ s.t. $v \in G_{r_v}$.

– A least common ancestor data structure over $\mathcal{T}$.

The space bottleneck is the size of the sets maintained. Each vertex v belongs to G_r° for $O(\lg n)$ nodes r of $\mathcal{T}$. For each of the $O(1)$ paths in the frame and separator of each such node r, v has a set of $O(\epsilon^{-1})$ connections. Hence the total space required by Thorup's oracle is $O(\epsilon^{-1} n \lg n)$.

We next describe how a query is performed. Given a u-to-w distance query, let r be the least common ancestor of r_u and r_w in $\mathcal{T}$. The algorithm computes, for each path Q of $S_r \cup F_r$ the length of a shortest u-to-w path that intersects Q using the connections $C(u, Q)$ and $C(w, Q)$ (of both type 0 and type 1). By construction of $\mathcal{T}$, the number of such paths Q is constant. It is easy to see that computing the distance estimate for each Q can be done in $O(\epsilon^{-1})$ time. Thus, an $(1 + \epsilon)$-approximate distance is produced in $O(\epsilon^{-1})$ time.

Efficient construction We now mention some, but not all the details of Thorup's $O(\epsilon^{-2} n \lg^3 n)$-time construction algorithm. Refer to [12, subsection 3.6] for the full details. The computation of the connections and connection lengths is done top-down the decomposition tree $\mathcal{T}$. Naively using Corollary 1 on G_r° for all $r \in \mathcal{T}$ is efficient, but only generates type-0 connections on S_r. Using Corollary 1 on G_r would produce type-0 connections on F_r, but is not efficient since $|F_r|$ can be much larger than $|G_r^\circ|$. Instead, For each path Q in F_r, the algorithm uses the reduction $\bar{Q}$ of Q to the vertices of Q that belong to G_r°. Let G_r^Q be the graph composed of G_r° and $\bar{Q}$. Note that $|G_r^Q| = O(|G_r^\circ|)$. The type-0 connections on F_r can now be computed by applying Corollary 1 to G_r^Q.

It remains to compute type-1 connections. Recall that these connection lengths reflect distances in the entire graph, not just in G_r. Clearly, applying Corollary 1 on G for every r is inefficient. Instead, the computation is done top-down $\mathcal{T}$ using an auxiliary construction. This construction augments G_r° with ϵ-covers of the separators of all ancestors of r in $\mathcal{T}$ with respect to the vertices of G_r°. These ϵ-covers have already been computed (type-0 connections at the ancestor), and represent distances outside G_r. Due to space constraints we defer the details to the next section, where we handle the more general case of vertex labels.

4 Undirected Approximate Vertex-Label Distance Oracle

The idea is to adapt Thorup's oracle (Section 3) to the vertex-label case. Thorup's oracle supports one-to-one (vertex-to-vertex) distance queries, whereas here we need one-to-many distance queries. Given two vertices u, v, Thorup's oracle finds the LCA of r_u and r_v in $\mathcal{T}$, and uses its connections to produce the answer. In a one-to-many query, there is no analogue for v. We do know, however, that a shortest u-to-λ path must intersect the separator of the leafmost node r in $\mathcal{T}$ that contains u and some λ-labeled vertex. The node r takes the role of the LCA of r_u and r_v. In order to be able to use r's connections in a distance query one must make sure that r's connections represent approximate distances to λ-labeled vertices in the entire graph, not just in G_r°.

We define a set $\mathcal{L}$ of new (artificial) vertices, one per label. For every $r \in \mathcal{T}$, let $\mathcal{L}_r = \{\lambda \in \mathcal{L} | G_r^\circ \cap V_\lambda \neq \emptyset\}$ be the restriction of $\mathcal{L}$ to labels in G_r°.

Simply connecting each vertex of V_λ to an artificial vertex representing the label λ is bound to fail. To see why, suppose vertices u and v both have label λ. Adding an artificial vertex λ and zero-length undirected edges $v\lambda$ and $u\lambda$ creates a zero-length path between u and v that does not exist in the original graph. While this does not change the distance between any vertex and its closest λ-labeled vertex, it may change distances between a vertex and its closest λ'-labeled vertex ($\lambda' \neq \lambda$). Therefore, we would have liked to add, for each label λ *separately*, a single artificial vertex λ, and compute the connection sets $C(\lambda, Q)$. Doing so would result in correct distance estimates, but is not efficient. We show how to compute the connections $C(\lambda, Q)$ without actually performing this inefficient procedure. Instead of having a single artificial vertex per label, it is split into many artificial vertices (one for each incident edge). The problem with this approach is that the number of connections becomes too large (each split vertex has its own set of $O(\epsilon^{-1})$ connections). We use an extension of the thinning procedure (Lemma 4) to select a small subset of these connections and still get the desired approximation.

Another point that we must address is that, for $\lambda \in \mathcal{L}_r$, the type-1 connections $C(\lambda, Q)$ should reflect the minimum distances between the connections of λ on Q to the closest vertex with label λ in G, not just to vertices with label λ in G_r°. We show how to achieve this by an extension of the auxiliary construction used to compute they type-1 connections in Thorup's unlabeled oracle.

We start with the extended thinning lemma.

Lemma 6. *Let $\{u_i\}$ be vertices and Q be a shortest path. Given ordered ϵ-covering sets $\{D(u_i, Q)\}$ it is possible to compute in linear time a clean and ordered 3ϵ-covering connections set C of size $O(\epsilon^{-1})$ which represent approximated distances from any $q \in Q$ to its closest vertex among $\{u_i\}$.*

Proof. We first convert every connection length $\ell(u_i, Q)$ to reflect an approximated length from q to its closest vertex $u^* \in \{u_j\}$, rather than to u_i. We obtain these lengths using the fact that q is ϵ-covered with respect to u^* by some connection in $D(u^*, Q)$. Let Z_u be the graph composed of the following. See Figure 3 in the appendix for an illustration.

1. $\bar{Q}$, the reduced form of Q to connections of all $\{D(u_i, Q)\}$.
2. vertices $\{u_i\}$, along with edges between each u_i to its connections, with lengths equal to the corresponding connection lengths.
3. vertex u, connected with zero-length edges to all $\{u_i\}$.

By the ϵ-covering property, the distances between every $q \in \bar{Q}$ and u in Z_u represent approximate distances between q and its closest vertex $u^* \in \{u_j\}$ in G. To see this, assume $q \in \bar{Q}$ is a connection of u_1, and is closest to u^*. Let q^* be a connection of u^* which ϵ-covers q w.r.t. u^*. Then $\delta_{Z_u}(q, u^*) \leq \delta_Q(q, q^*) + \ell(q^*, u^*) = \delta_G(q, q^*) + \ell(q^*, u^*) \leq (1+\epsilon)\delta_G(q, u^*)$.

It is possible to compute all shortest paths from u in Z_u in linear time; first, relax all edges incident to u and $\{u_i\}$. Then, relax the edges of $\bar{Q}$ by going first

in one direction along Q and then relaxing the same edges again in the other direction. For connection p on $\bar{Q}$, a u-to-p shortest path first reaches Q along one of $\{u_i\}$ edges and then walks along Q toward p. Hence the relaxation was done in the correct order. We update the connection lengths to the distances thus computed.

Let $\tilde{D}(u,Q)$ denote the ordered union of all connections, along with the updated connection lengths. Since all $\{D(u_i,Q)\}$ were ordered, it is possible to order their union in linear time. Let G_u be the graph obtained from G by adding an apex u connected with zero length edges to all $\{u_i\}$. We stress that G_u is not constructed by the algorithm, but only used in the proof. $\tilde{D}(u,Q)$ is an ϵ-cover of Q with respect to u in G_u. Now apply Lemma 4 to $\tilde{D}(v,Q)$ with $\epsilon_0 = \epsilon_1 = \epsilon$ to obtain a 3ϵ-cover of Q with respect to u in G_u of size $O(\epsilon^{-1})$. □

4.1 Vertex-label distance oracle for undirected graphs

The vertex labeled distance oracle is very similar to the unlabeled one (Section 3). It uses the same decomposition tree $\mathcal{T}$, and stores, for each $r \in \mathcal{T}$, the same covering sets. The only difference is that, in addition to the covering sets $C(u,Q)$ for each vertex $u \in G_r^\circ$, the oracle also stores connection information for labels as we now explain.

For every $r \in \mathcal{T}$ and $\lambda \in \mathcal{L}_r$, the oracle stores connections $C(\lambda,Q)$ of both type-0 and type-1. The type-0 connections $C(\lambda,Q)$ are connections in the graph obtained from G_r° by adding an artificial vertex λ, along with zero length edges from all λ-labeled vertices in G_r° to λ. The type-1 connections $C(\lambda,Q)$ are connections in the graph obtained from G by adding an artificial vertex λ, along with zero length edges from all λ-labeled vertices in G to λ. Before explaining how to compute these connections we discuss how a distance query is performed.

Obtaining the distance from u to λ is done by finding the lowest ancestor r of r_u with $\lambda \in \mathcal{L}_r$. A shortest u-to-λ path must cross S_r, and perhaps also F_r. The algorithm estimates, for each path Q of $S_r \cup F_r$, the length of a shortest u-to-λ path that intersects Q, using the connections $C(u,Q)$ and $C(\lambda,Q)$ stored for r (Since $\lambda \in \mathcal{L}_r$, r does store Q-to-λ connections).

Finding r can be done by binary search on the path from r_u to the root of $\mathcal{T}$. The number of steps of the binay search is $O(\lg\lg n)$. Finding whether a node r' has a vertex with label λ can be done, e.g., by storing all unique labels in $G_{r'}^\circ$ in a binary search tree, or by hashing. In the former case finding r takes $O(\lg\lg n \lg|L|)$ time, and in the latter $O(\lg\lg n)$, assuming the more restrictive word-RAM model of computation.

It remains to show how the connections are computed. We begin with the type-0 connections. For every $r \in \mathcal{T}$, for every $Q \in F_r \cup S_r$, the algorithm computes ordered ϵ-covering sets of connections on Q w.r.t. each vertex of G_r° to Q by invoking Corollary 1 to G_r°. This takes $O(\epsilon^{-1}|V(G_r^\circ)|\lg n)$ time (using [5] for shortest path computation). For each $\lambda \in \mathcal{L}_r$, let n_λ denote the number of λ-labeled vertices in G_r°. The total number of connections to λ-labeled vertices in G_r° is $O(\epsilon^{-1}n_\lambda)$. The algorithm next applies the extended thinning lemma

(Lemma 6) to get a connections set $C(\lambda, Q)$ of size $O(\epsilon^{-1})$ in $O(\epsilon^{-1} n_\lambda)$ time. Since $\sum_\lambda n_\lambda = O(|V(G_r^\circ)|)$, the runtime for a single r and Q is $O(\epsilon^{-1}|V(G_r^\circ)|)$.

We now show how to compute the type-1 connections without invoking Corollary 1 to the entire input graph G at every call.

Lemma 7. *Let $r \in \mathcal{T}$. Type-1 connections for r can be computed using just the (type-0) connections of strict ancestors of r. Computing all type-1 connections for all $r \in \mathcal{T}$ can be done in $O(\epsilon^{-2} n \lg^3 n)$ time.*

Proof. Let Q be a path in F_r. Let X_r^Q be the graph composed of the following: (see Figure 4 in the appendix for an illustration)

- The vertices $\mathcal{L}_r$
- The vertices and edges of $\bar{Q}$, the reduction of Q to $V(Q) \cap V(G_r^\circ)$.
- For each strict ancestor r' of r, for each path $Q' \in S_{r'}$, the vertices and edges of $\bar{Q}'$, the reduction of Q' to vertices that are (type-0) connections (in $G_{r'}^\circ$)) of Q' w.r.t. vertices in $Q \cup \mathcal{L}_r$, along with edges representing the corresponding connection lengths.

The algorithm creates a graph $\hat{X}_r^Q$ from X_r^Q by breaking every artificial vertex λ in X_r^Q into many copies $\{\lambda_e\}$, one per incident edge of λ. We stress that the artificial vertices λ_e are not directly connected to each other in $\hat{X}_r^Q$. Hence, the problem of shortcuts mentioned earlier is avoided. See Figure 5 in the appendix for an illustration.

Note that splitting vertices in this way does not increase the number of edges in the $\hat{X}_r^Q$. The algorithm applies Corollary 1 to $\hat{X}_r^Q$ and Q, obtaining a small sized ϵ-cover $C(\lambda_e, Q)$ for every λ_e.

Let q be any vertex of $\bar{Q}$, and let λ be a label in G_r°. Let P be a shortest q-to-λ path in G. Let r' be the rootmost strict ancestor of r such that $S_{r'}$ is intersected by P. Note that r' must exist since $q \in F_r$, so q belongs to the separator of some strict ancestor of r. Thus P is entirely contained in $G_{r'}^\circ$. Let Q' be a path in $S_{r'}$ intersected by P. By construction of $\hat{X}_r^Q$, it contains an ϵ-covering set of connections of Q' with respect to q in $G_{r'}^\circ$, as well as the edges of $\bar{Q}'$ and an ϵ-covering set of connections of Q' with respect to λ in $G_{r'}^\circ$. Hence, by Lemma 2, there exists a shortest q-to-λ_e path (for some artificial vertex λ_e) in $\hat{X}_r^Q$ whose length is at most $(1+\epsilon)$ times the length of P. On the other hand, because the vertices λ_e (for any $\lambda \in \mathcal{L}_r$) are not directly connected to each other in $\hat{X}_r^Q$, every path in $\hat{X}_r^Q$ corresponds to some path in G, so shortest paths in $\hat{X}_r^Q$ are at least as long as those in G. This proves that $\hat{X}_r^Q$ correctly represents all desired type-1 connection lengths.

We proceed with describing the construction of the connection sets of the appropriate sizes. To bound the size of the connections $\{C(\lambda_e, Q)\}$, we count the number of edges incident to λ in X_r^Q (i.e., before it is split). There is an edge for each of the $O(\epsilon^{-1})$ connections of λ on each of the $O(\lg n)$ paths of separators of ancestors of r. For each such edge there is a vertex λ_e with an ϵ-covering set of $\bar{Q}$ of size $O(\epsilon^{-1})$. Thus, the total number of connections of $\bar{Q}$ for all λ_e vertices is $O(\epsilon^{-2} \lg n)$. The algorithm applies Lemma 6, the extended

thinning procedure, to $\{C(\lambda_e, Q)\}_e$ to get $C(\lambda, Q)$ of size $O(\epsilon^{-1})$. Doing so for all labels in G_r° requires $O(\epsilon^{-2}\lg n + \epsilon^{-1}|\mathcal{L}_r|)$ space.

We now bound the running time. Since splitting vertices does not increase the number of edges, applying Corollary 1 to $\hat{X}_r^Q$ takes $O(\epsilon^{-2}|V(G_r^\circ)|\lg^2 n)$ time. Applying Lemma 6 is done within the same time bound. To conclude, the total runtime over all nodes of $\mathcal{T}$ is $O(\epsilon^{-2} n \lg^3 n)$. □

We have thus established our main theorem:

Theorem 1. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n)_{space}\ , O(\lg\lg n + \epsilon^{-1})_{time}\rangle$ Vertex-Label Distance Oracle can be constructed within $O(\epsilon^{-2} n \lg^3 n)$ time w.h.p.* [7] *in an undirected planar graph with n vertices.*

References

1. S. Baswana, A. Gaur, S. Sen, and J. Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *35th ICALP*, pages 609–621, 2008.
2. S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *47th FOCS*, pages 591–602, 2006.
3. S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.
4. S. Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *20th ESA*, pages 325–336, 2012.
5. M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.
6. D. Hermelin, A. Levy, O. Weimann, and R. Yuster. Distance oracles for vertex-labeled graphs. In *38th ICALP*, pages 490–501, 2011.
7. K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *38th ICALP*, pages 135–146, 2011.
8. K. Kawarabayashi, C. Sommer, and M. Thorup. More compact oracles for approximate distances in undirected planar graphs. In *24th SODA*, pages 550–563, 2013.
9. P. N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *13th SODA*, pages 820–827, 2002.
10. M. Li, C. C. C. Ma, and L. Ning. $(1+\epsilon)$-distance oracles for vertex-labeled planar graphs. In *10th TAMC*, pages 42–51, 2013.
11. R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, pages 177–189, 1979.
12. M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.
13. M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.
14. C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *23rd SODA*, pages 202–208, 2012.

[7] The probability in the construction time is only due to the use of perfect hashing.

Appendix

A Proof of Lemma 4

Lemma 4. *Let Q be a path in an undirected graph, and let v be a vertex. Let $D(v,Q)$ be an ordered ϵ_0-cover of Q w.r.t. v. For any $\epsilon_1 \le 1$, a clean and ordered $(2\epsilon_0+\epsilon_1)$-cover $C(v,Q) \subseteq D(v,Q)$ of size $O(\epsilon_1^{-1})$ can be constructed in $O(|D(v,Q)|)$ time.*

Proof. The proof is constructive. Let $(\bar{q},v)$ be a connection with minimal connection length in $D(v,Q)$. The vertex $\bar{q}$ splits Q into two subpaths, Q_0 and Q_1 . For each $Q^{'} \in \{Q_0,Q_1\}$, the algorithm operates as follows. First, it adds $(\bar{q},v)$ to $C(Q^{'},v)$. The algorithm will now progress towards the other endpoint of $Q^{'}$. We say $\tilde{q}$ *semi ϵ-covers* q^* if $\delta_Q(q^*,\tilde{q})+\ell(\tilde{q},v) \le (1+\epsilon)\ell(q^*,v)$. [8]

Let $(\tilde{q},v)$ be the last connection added to $C(Q^{'},v)$. Let (q^*,v) be the next connection of $D(v,Q)$ that has not been considered yet. The algorithm adds (q^*,v) unless $\tilde{q}$ already semi ϵ_1-covers (q^*,v). The algorithm returns $C(v,Q)=C(Q_0,v)\cup C(Q_1,v)$.

We first prove that $C(v,Q)$ is a $(2\epsilon_0+\epsilon_1)$-cover. Let q be a vertex in Q. Let d be the connection in $D(v,Q)$ which ϵ_0-covers q. Let c be a connection of $C(v,Q)$ that semi ϵ_1-covers d (it might be that $c=d$).

We know that

$$\delta(q,c) \le \delta(q,d)+\delta_Q(d,c) \text{ (triangle inequality)} \tag{2}$$

$$\delta_Q(d,c)+\ell(c,v) \le (1+\epsilon_1)\ell(d,v) \text{ (c semi ϵ_1-covers d)} \tag{3}$$

$$\delta(q,d)+\ell(d,v) \le (1+\epsilon_0)\delta(q,v) \text{ (d ϵ_0-covers q)} \tag{4}$$

We have that: $\delta(q,c)+\ell(c,v) \underset{(2)}{\le} \delta(q,d)+\delta_Q(d,c)+\ell(c,v) \underset{(3)}{\le} \delta(q,d)+(1+\epsilon_1)\ell(d,v) \le (1+\epsilon_1)(\delta(q,d)+\ell(d,v)) \underset{(4)}{\le} (1+\epsilon_1)(1+\epsilon_0)\delta(q,v) = (1+\epsilon_0+\epsilon_1+\epsilon_0\epsilon_1)\delta(q,v) \underset{\epsilon_1\le 1}{\le} (1+(2\epsilon_0+\epsilon_1))\delta(q,v)$, and the approximation bound follows.

We now turn to show the generated cover is of $O(\epsilon_1^{-1})$ size. For $Q^{'} \in \{Q_0,Q_1\}$, we show it is of size $O(\epsilon_1^{-1})$. Let $\{c_i\}_{i\ge 1}$, of size k, be the chosen connections along $Q^{'}$, numbered by their order along $Q^{'}$ toward the other endpoint t of $Q^{'}$, starting with $c_1=\bar{q}$. We examine the function $f(c_i)=\delta_Q(t,c_i)+\ell(c_i,v)$. We observe that $f(c_i)-f(c_{i+1}) = (\delta_Q(t,c_i)+\ell(c_i,v))-(\delta_Q(t,c_{i+1})+\ell(c_{i+1},v)) = \ell(c_i,v)+\delta_Q(c_i,c_{i+1})-\ell(c_{i+1},v) \ge \epsilon_1\ell(c_{i+1},v) \ge \epsilon_1\ell(\bar{q},v) \ge \epsilon_1\delta(\bar{q},v)$. Thereby, $f(c_{i+1}) \le f(c_1)-i\epsilon_1\delta(\bar{q},v)$, hence $f(c_k) \le f(c_1)-(k-1)\epsilon_1\delta(\bar{q},v)$. Note that $f(c_k)=\delta_Q(t,c_k)+\ell(c_k,v) \ge \delta(t,v) \ge \delta_Q(t,\bar{q})-\delta_Q(\bar{q},v)$. Using the lower and upper bounds over $f(c_k)$, we have that $\delta_Q(t,\bar{q})-\delta(\bar{q},v) \le f(c_k) \le f(c_1)-(k-1)\epsilon_1\delta(\bar{q},v) = \delta_Q(t,\bar{q})+\ell(\bar{q},v)-(k-1)\epsilon_1\delta(\bar{q},v) \le \delta_Q(t,\bar{q})+(1+\epsilon_0)\delta(\bar{q},v)-(k-1)\epsilon_1\delta(\bar{q},v)$. Hence $((k-1)\epsilon_1-(1+\epsilon_0)\delta(\bar{q},v) \le \delta_Q(\bar{q},v)$ and so $k \le 1+\frac{2+\epsilon_0}{\epsilon_1}$. Therefore the size of the connections obtained over both $\{Q_0,Q_1\}$ is $O(\epsilon_1^{-1})$. ◻

[8] The semi ϵ-cover definition is similar to ϵ-cover definition. The only difference is that $\delta(q^*,v)$ was replaced by $\ell(q^*,v)$ for fast computation purposes.

B A flaw in Thorup's treatment of the undirected case

There is another notion of covering, apart from ϵ-covering and quasi-ϵ-covering [12].

Definition 3. q^* strictly ϵ-covers q *w.r.t.* v *if* $\delta(q, q^*) + \ell(q^*, v) \leq \delta(q, v) + \epsilon\delta(v, Q)$.

In [12], Thorup uses quasi-ϵ-covers and strict-ϵ-covers, but does not use (plain) ϵ-covers.[9] Most of the discussion in [12] is devoted to the directed case, which uses yet another notion of covering. When treating the undirected case, Thorup claims that all lemmas, except for the efficient construction procedure, carry over from the directed case to the undirected case when the directed definition of ϵ-covering is replaced with strict ϵ-covering. The treatment of the efficient construction for the undirected case is more detailed, where a procedure for efficiently constructing quasi-ϵ-covering sets is given (Lemma 3, [12, Lemma 3.18]).

We believe that the treatment of the undirected case in [12] suffers from two flaws. First, the proof of the thinning procedure does not seem to carry over from the directed case to the undirected case when using strict ϵ-covers. Second, since the construction is of quasi-ϵ-covers, whereas all other parts of the undirected oracle in [12] assume strict-ϵ-covers, the correctness of the entire oracle is not established.

Our algorithm does not use strict ϵ-covers at all. We use Thorup's efficient construction of quasi ϵ-covers, which, by Proposition 1 is also a $O(\epsilon)$-cover, and prove that the thinning procedure and query algorithm work for ϵ-covers.

C Vertex-Label distance oracle for directed planar graphs

Thorup shows that the problem of constructing a distance oracle for a directed graph can be reduced to constructing a distance oracle for a restricted kind of graph, defined in the following.

Definition 4. *A set T of arcs in a graph H is a (t, α)-layered spanning tree if it satisfies the following properties:*

- *Disoriented - it can be oriented to form a spanning tree of H.*
- *Each branch (a path from the root of T) is a concatenation of at most t shortest paths in H.*
- *Each shortest path in a branch of T is of length at most α*

Definition 5. *A graph H is called (t, α)-layered if it has a (t, α)-layered spanning tree.*

Definition 6. *A scale-(α, ϵ) distance oracle for a (t, α)-layered graph H is a data structure that, when queried for $\delta_H(v, w)$ returns*

$$d(v, w) = \begin{cases} d \in [\delta_H(v, w), \delta_H(v, w) + \epsilon\alpha] & \delta_H(v, w) \leq \alpha \\ \infty & \textit{otherwise} \end{cases}$$

[9] Thorup did not use the terms strict and quasi.

The reduction is summarized in the following lemma. Let N be the maximum length of an arc in G. [10]

Lemma 8. *([12, Section 3.1,3.2,3.3]) Given a scale-(α, ϵ') $\langle O(s(n,\epsilon'))_{space}$, $O(t(\epsilon'))_{time}\rangle$ algorithm, a $(1{+}\epsilon)$-stretch $\langle O(s(n,\epsilon)\lg(nN))_{space}$, $O(t(\frac{1}{4})\lg\lg(nN) + t(\frac{\epsilon}{4}))_{time}\rangle$ algorithm can be constructed to any input graph.*

Thorup shows each graph can be decomposed to a $(3,\alpha)$-layered graphs, of total linear size (with α the distance bound of the graph). Therefore its suffices to show how to construct a scale-(α,ϵ) distance oracle.

To do so, Thorup shows a directed variant to Lemma 1:

Lemma 9. *Let Q be a shortest path in a directed graph H. There exist sets $C(u,Q)$ of $O(\epsilon^{-1})$ vertices of Q for all $u \in H$, where:*

1. *$C(u,Q)$ are called the* connections of u on Q.
2. *The distance between u and a connection $q \in C(u,Q)$ is called the* connection length of u and q.
3. *For every $u,w \in H$, if a shortest u-to-w path in H intersects Q, then $\delta_{H_Q^{uw}}(u,w) \le \delta_H(u,w) + \epsilon\alpha$.*
   Here H_Q^{uw} is the graph with vertices u,w, and the vertices of the reduction of Q to $C(u,Q) \cup C(w,Q)$, and with u-to-Q and Q-to-v arcs whose lengths are the corresponding connection lengths of $C(u,Q)$ and $C(w,Q)$.

The ϵ-covering definition in the directed case is not the same as in the undirected. It uses a $O(\epsilon\alpha)$ additive error to the approximation rather $\epsilon\delta_H(u,w)$. Moreover, the u-to-Q cover and u-from-Q cover are different (because of the directedness) but obtained in similar manner.

As in the undirected case, given a $(3,\alpha)$-layered graph, the algorithm keeps a recursive graph decomposition $\mathcal{T}$. The distance oracle keeps, for each $r \in \mathcal{T}$, for each vertex $u \in G_r^\circ$:

1. connections $C(u,Q)$ of type-0 for all $Q \in F_r$.
2. connections $C(u,Q)$ (Q to u distances) of type-1 for all $Q \in F_r$.
3. connections $C(u,Q)$ (separate Q to u and u to Q distances) of type-0 for all $Q \in S_r$.

A u-to-w query is done similar to the undirected case, using their relevant connections found in the LCA of r_u and r_w. The construction of the connections is similar to the undirected case; type-0 connections are computed in G_r° where the frame is reduced to the vertices of G_r° (using Lemma 1). type-1 connections of node $r \in \mathcal{T}$ are computed in a graph augmented by type-0 connections of ancestors of r (see [12, Section 3.6] and Lemma 7 of the undirected case).

For the vertex-label case, for any $r \in \mathcal{T}$ we define $\hat{G}_r^\circ$ be G_r° along with vertices $\mathcal{L}_r$ and arcs from any λ-labeled vertex to λ for each vertex $\lambda \in \mathcal{L}_r$.

The vertex-label distance oracle keeps, for each $r \in \mathcal{T}$, for each vertex $u \in \hat{G}_r^\circ$ (either in G_r° or artificial vertex from $\mathcal{L}_r$):

[10] nN is an upper bound on $\delta_G(\cdot)$.

1. connections $C(u, Q)$ of type-0 for all $Q \in F_r$.
2. connections $C(u, Q)$ (Q to u distances) of type-1 for all $Q \in F_r$.
3. connections $C(u, Q)$ (separate Q to u and u to Q distances) of type-0 for all $Q \in S_r$.

A u-to-λ query is done similarly to the vertex-to-vertex case relatively to the u and λ connections of the lowest ancestor of r_u with $\lambda \in \mathcal{L}_r$. The construction is similar to the vertex-to-vertex construction, but instead done over $\hat{G}_r^\circ$. See Figure 6 for the type-1 connections construction directed variant.

We thus get the following theorem:

Theorem 2. *A $(1+\epsilon)$-stretch $\langle O(\epsilon^{-1} n \lg n \lg(nN))_{space}$, $O(\lg\lg n \lg\lg(nN) + \epsilon^{-1})_{time}\rangle$ Vertex-Label Distance Oracle can be constructed in $O(\epsilon^{-2} n \lg^3 n \lg(nN))$ time w.h.p.* [11] *for a directed planar graph with n vertices and maximum arc length N.*

[11] The probability in the construction time is only due to the use of perfect hashing.

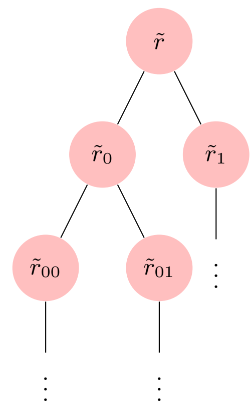

Fig. 1: An illustration of the decomposition tree $\mathcal{T}$. The root $\tilde{r}$ is associated with $G_{\tilde{r}} = G$. The children of $\tilde{r}$, $\tilde{r_0}$ and $\tilde{r_1}$, are associated with the interior and exterior of the separator $S_{\tilde{r}}$ of $G_{\tilde{r}}$.

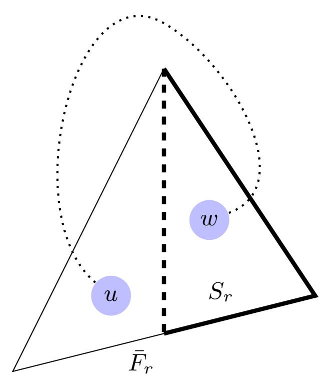

Fig. 2: The solid lines (thin and thick) indicate $\bar{F}_r$, the reduced frame of G_r. The bold lines (solid and dashed) indicate S_r, the separator of G_r. Vertices u and w are vertices of G_r° separated by S_r. Every u-to-w path must intersect S_r. The dashed line shows a possible shortest u-to-w path in G.

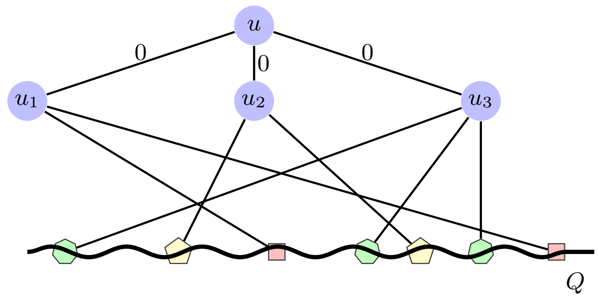

Fig. 3: Illustration of the situation in the proof of the extended thinning lemma (Lemma 6). Each vertex in $\{u_i\}$ has different connections on Q. The distance from u_1 to any connection of u_2 is approximated using the connections of u_1.

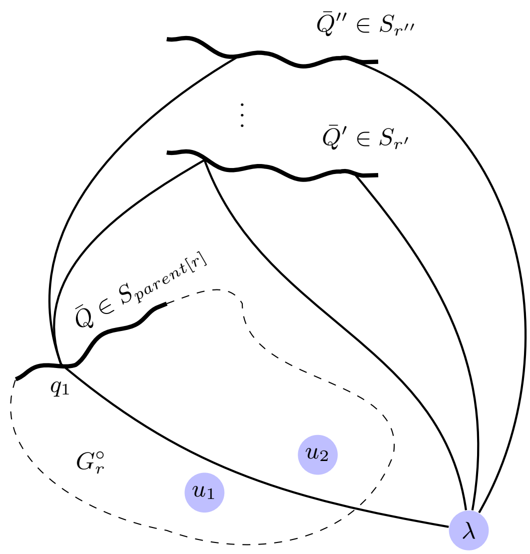

Fig. 4: The figure illustrates a part of X_r^Q. In this example Q is a path in the separator of the parent of r (in general Q is a path in F_r, so it may belong to the separator of an ancestor of r). The vertices of G_r° are enclosed in F_r, which is represented by the dashed lines and by $\bar{Q}$. The vertices u_1 and u_2 are λ-labeled vertices of G_r°, and are not part of X_r^Q. Paths from $\bar{Q}$ to λ-labeled vertices such as u_1 and u_2 are represented in X_r^Q by edges between $\bar{Q}$ and λ. These edges correspond to the type-0 connections of λ on $\bar{Q}$ in the parent of r. All solid edges are part of X_r^Q. A shortest path from $q_1 \in \bar{Q}$ to $\lambda \in \mathcal{L}_r$ is approximated by connections from q_1 to a separator of an ancestor of r and from there to λ. Note that $C(\lambda, Q')$ represent distances from Q' to a λ-labeled vertex which is not necessarily in G_r°.

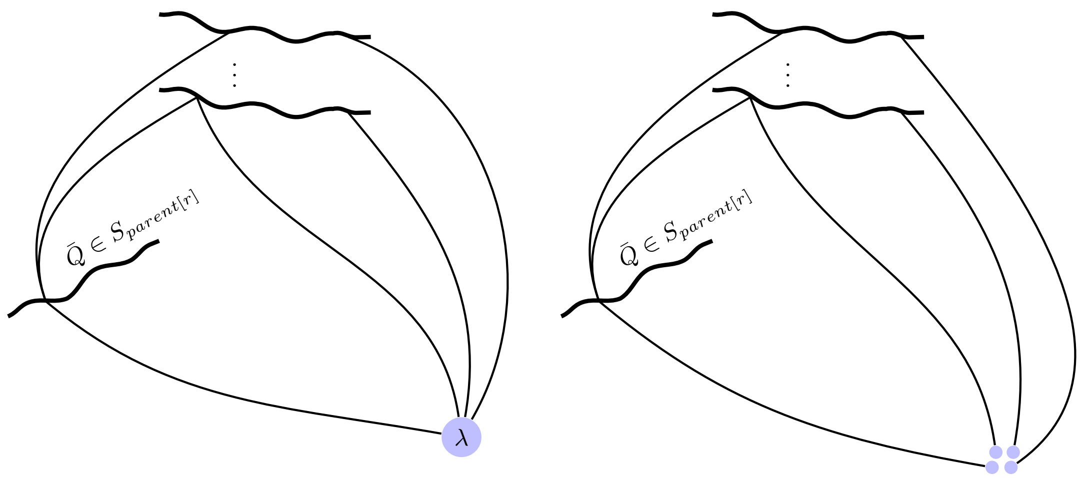

Fig. 5: Illustration of the utility of splitting an artificial vertex λ. On the left (X_r^Q) undesired shortcuts (teleportation) might occur. On the right ($\hat{X}_r^Q$) teleportation does not occur.

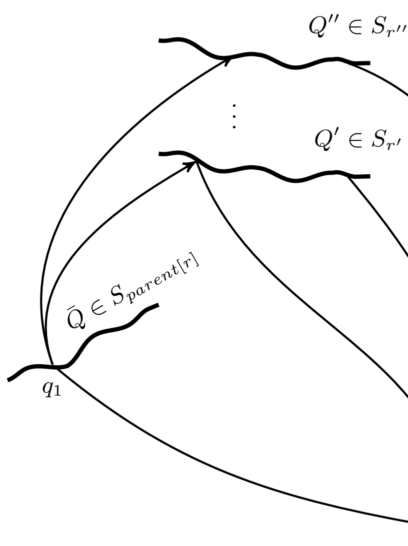

Fig. 6: Illustration of part of the auxiliary graph X_r^Q for the directed case. No teleportation can occur because each artificial vertex λ has only incoming arcs, and no outgoing arcs.